REMARKS

Claims 1, 4, 11, 19, 21, 24, 28, 33, 35, 36, 39-42, 47, 51, 53, 56, 57, and 60 have been amended. Claims 63-66 have been added. Therefore claims 1-66 remain pending in the application. Reconsideration is respectfully requested in light of the following remarks.

Section 112, Second Paragraph, Rejection:

The Office Action rejected claims 11, 19, 24, 28, 35, 51 and 60, under 35 U.S.C. § 112, second paragraph, as being indefinite for containing trademarks. These claims have been amended to refer to respective components being based on a specific technical specification, as opposed to a trademark name. Thus, Applicants assert that amended claims 11, 19, 24, 28, 35, 51 and 60 are not indefinite and removal of the 35 U.S.C. § 112, second paragraph, rejection is respectfully requested.

Section 102(b) Rejection:

The Office Action rejected claims 1-62 under 35 U.S.C. § 102(b) as being anticipated by Liang, et al. "Dynamic Class Loading in Java Virtual Machine," 1998 ACM (hereinafter "Liang").

Regarding claim 1, Applicants assert that Liang fails to teach a class loader, in response to an invocation from a class loader controller, loading a class in the application, wherein the class loader is one of a hierarchal stack of class loaders each configured to load one or more classes in the application, wherein the class loader controller provides an interface to the hierarchal stack of class loaders and a common entry point for loading classes of the application, as recited in Applicants' amended claim 1.

Liang teaches class loaders for dynamically loading software components on the Java platform, (Liang, Abstract). Figure 2, on page 37 of Liang, shows a number of

applets and an application class loader delegating the loading of system classes to a system class loader. (Liang, page 37, Figure 2, and page 38, 2nd paragraph). Liang fails to mention anything regarding a class loader controller and additionally fails to disclose a class loader loading a class in the application in response to an invocation from a class loader controller that provides an interface to the hierarchal stack of class loaders and a common entry point for loading classes of the application. In contrast, Liang teaches, "[t] Java virtual machine will use [class loader] L to load classes referenced by [class] C" (Liang, page 37, section 2.1, paragraph 2). Further, Liang teaches that class files contain symbolic references to other classes and that such symbolic references are resolved at link time by the Java virtual machine. Liang also teaches that "to resolve a symbolic reference to a class, the Java virtual machine must load the class file and create the class type" (Liang, page 37, paragraph 5). Thus, Liang is teaching that a Java Virtual machine uses class loaders to load classes in response to references in class files and does not teach the loading of classes in response to an invocation from a class loader controller that provides an interface to the hierarchal stack of class loaders and a common entry point for loading classes of the application.

Furthermore, Liang does not disclose wherein the class loader is one of a hierarchal stack of class loaders each configured to load one or more classes in the application. The Examiner cites Figure 2, and section 2.1 of Liang. Applicants note, however, that Figure 2 of Liang illustrates how one class loader can delegate the loading of a class to another class loader, such as delegating the loading of Java system classes to a system class loader. Figure 2 of Liang does not illustrate a hierarchal stack of class loaders. Section 2.1 of Liang gives an overview of class loading in Java, but Applicants can find no mention of a hierarchal stack of class loaders in section 2.1 of Liang. In contrast, as shown above, section 2.1 describes how the Java virtual machine uses a class' defining loader to load classes referenced by that class. Liang is silent regarding a hierarchal stack of class loaders.

Moreover, Liang fails to teach a class loader controller that provides an interface to the hierarchal stack of class loaders and a common entry point for loading classes of

the application. Instead, Liang discloses that each classes defining class loader is used by the Java virtual machine, in response to references in the code, to load additional classes. Hence, instead of teaching a class loader controller that provides an interface to a hierarchal stack of class loaders, Liang teaches individually interfaces each loader as necessary to resolve class references. Further, Applicants can find no reference in Liang to a class loader controller providing a common entry point for loading classes of the application.

Thus, in light of the above remarks, Applicants assert that the rejection of claim 1 is not supported by the cited art and withdrawal of the rejection is respectfully requested. Similar remarks as discussed above in regard to claim 1 apply to claims 21, 36 and 53.

Regarding claim 5, Liang fails to teach registering the loaded class with a dirty class monitor; and the dirty class monitor performing said detecting the class has been changed. The Examiner contends that such teaching is inherent in Figure 3 of Liang. Applicants disagree. Liang specifically teaches that Figure 3 illustrates "how a Server class can dynamically redirect the service requests to a new version of the Service class" (Liang, section 3.1, paragraph 4). Neither Figure 3, nor the accompanying text, discloses anything, either explicitly or inherently, regarding registering a loaded class with a dirty class monitor or regarding the dirty class monitor detecting that the class has been changed. In fact, Applicants can find no teaching in Liang regarding how a new version of a class is detected.

Thus, in light of the above remarks, Applicants assert that the rejection of claim 5 is not supported by the cited art and withdrawal of the rejection is respectfully requested. Similar remarks as discussed above in regard to claim 5 apply to claim 6.

Regarding claim 21, in addition to the arguments presented above regarding claim 1, Applicants also submit that Liang fails to teach the class loader controller replacing one or more class loaders for one or more classes with dependencies on the changed class, as recited by Applicants amended claim 21. Liang teaches only "reloading a subset

of the classes already loaded in a running virtual machine" to prevent having to shutdown and restart an application. (Liang, page 39, section 3.1, second paragraph). Applicants can find no mention in Liang regarding a class loader controller replacing class loaders with dependencies on the changed class. Any interpretation of Liang to include a class loader controller replacing class loaders with dependencies on the changed class is merely speculation and hindsight analysis on the Examiner's behalf.

Thus, in light of the above remarks, Applicants assert that amended claim 21 further distinguishes over the prior art and that the rejection of claim 21 is not supported its withdrawal is respectfully requested.

Regarding claim 29, Applicants assert that Liang does not teach a system, wherein the program instructions implement an application server operable to provide access to the plurality of applications to clients of the application server, wherein one or more of the plurality of applications each includes a dynamic class reloading module comprising a hierarchical stack of class loaders.

Applicants note that the Examiner states, "claim 29 recites a system that has the claim functionality corresponding to the limitation recited in claim 1." Applicants submit, however, that Applicants' claim 29 is of a different scope than claim 1 and that Liang fails to teach wherein the program instructions implement an application server. Nor does Liang teach wherein the application server is operable to provide access to the plurality of applications to clients of the application server. Applicants can find no teaching or mention in Liang of an application server operable to provide access to a plurality of applications to clients of the application server, as recited by Applicants claim 29. Liang only discloses the use of class loaders in applications, such web browsers. (Liang, page 46, section 1, first paragraph, Figure 2, Section 2.2 second paragraph).

Further, Liang fails to teach wherein one or more of the plurality of applications each includes a dynamic class reloading module comprising a hierarchical stack of class loaders, wherein the hierarchical stack of class loaders includes a separate class loader for

each module in the particular application, and wherein each class loader is operable to reload one or more classes used by the particular application.

Additionally, Applicants assert that Liang fails to disclose "the replaced one or more class loaders reloading the one or more classes in the application with dependencies on the changed class" as recited in Applicants' claim 29. As discussed above regarding claim 21, Liang teaches only "reloading a subset of the classes already loaded in a running virtual machine" to prevent having to shutdown and restart an application. (Liang, page 39, section 3.1, second paragraph).

Thus, in light of the above remarks, Applicants assert that the rejection of claim 29 is not supported by the cited art and withdrawal of the rejection is respectfully requested.

Applicants also assert that numerous ones of the dependent claims recite further distinctions over the cited art. However, since the independent claims have been shown to be patentably distinct, a further discussion of the dependent claims is not necessary at this time.

Added Claims:

Regarding Applicants' new claims 63 – 66, Liang does not teach a hierarchal stack of class loaders each configured to load one or more classes in the application; a class loader controller configured to provide an interface to a hierarchal stack of class loaders and a common entry point for loading classes of the application; a dirty class monitor operable to detect that a class used by the application has been changed; and notify the class loader controller that the class has been changed.

CONCLUSION

Applicants submit the application is in condition for allowance, and notice to that effect is respectfully requested.

If any extension of time (under 37 C.F.R. § 1.136) is necessary to prevent the above referenced application from becoming abandoned, Applicants hereby petition for such extension. If any fees are due, the Commissioner is authorized to charge said fees to Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C. Deposit Account No. 501505/5181-91701/RCK.

Also enclosed herewith are the following items:

☒ Return Receipt Postcard

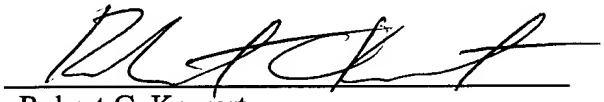
☐ Petition for Extension of Time

☐ Notice of Change of Address

☐ Fee Authorization Form authorizing a deposit account debit in the amount of $ for fees ( ).

☒ Other: Information Disclosure Statement, Form PTO-1449 and references A1-A6

Respectfully submitted,



Robert C. Kowert
Reg. No. 39,255
ATTORNEY FOR APPLICANT(S)

Meyertons, Hood, Kivlin, Kowert, & Goetzel, P.C.
P.O. Box 398
Austin, TX 78767-0398
Phone: (512) 853-8850

Date: July 13, 2004